SysCall Wrappers

CS3500 Course Project

Group 3

Week-3 Report

Process Wrappers

Aditya Jain, Aditya Srivastava, Harsh Vardhan Daga, Shreyanshu Gurjar

- Deadlock Detection Wrapper
  - System Call Hooking:
    - Intercepts pthread_mutex_lock(), pthread_mutex_unlock(), sem_wait(), sem_post().
    - Tracks interactions with mutexes & semaphores. Stops process if deadlock detected.
  - Deadlock Detection Across Threads and Processes:
    - Detects deadlocks caused by circular dependencies between threads and processes.
    - Identifies both mutex and semaphore deadlocks, where threads block each other indefinitely.
  - Cycle Detection:
    - Uses a resource graph to detect cycles, indicating deadlock and logs full deadlock cycle.
  - Logging:
    - Logs resource interactions (e.g., mutexes, semaphores) in hold and wait files.
    - Captures thread/process IDs and resource memory addresses involved in deadlocks.
- Zombie Process Logger
  - This wrapper extends the kill() and waitpid() system calls to ensure proper process termination and prevent the creation of zombie processe.
  - Ensures proper termination and cleanup of child processes while logging detailed information about terminated and zombie processes for easier debugging.

Process Wrappers

Aditya Jain, Aditya Srivastava, Harsh Vardhan Daga, Shreyanshu Gurjar

- IPC Logging Wrapper
  - System Call Hooking:
    - Uses dlsym(RTLD_NEXT, "<syscall>") to intercept IPC system calls like read(), write(), msgsnd(), semop(), kill() to eavesdrop on communications through pipes, message queues, shared mem.
  - Data Logging:
    - Logs IPC details, including message content (up to a set size) and operation parameters.
  - Performance Monitoring:
    - Captures timestamps, operation success/failure, and message size to aid in debugging, performance analysis, and performance bottlenecks.
  - Security Auditing:
    - Monitors IPC calls for unauthorized access, irregular behavior or security breaches.
- Priority Enforcement Wrapper
  - The Priority Enforcement Wrapper extends the fork() system call to set CPU and I/O priorities for child processes. This helps ensure that high-priority tasks receive sufficient resources while low-priority tasks don't hinder overall system performance
  - Adjusts the **CPU scheduling priority** using the **setpriority** system call.
  - **I/O priority** is set using a system command with **ionice**.

File I/O Wrappers

Yashvardhan, Kritang, Daksh

- Buffer
  - Implements a **buffering mechanism** to temporarily store data before writing it to a file. This can **improve performance** by reducing the number of write operations.
  - **buffer_init** : Initializes the buffer by setting its size to 0 and associating a file descriptor with it.
  - **buffer_add** : Adds data to the buffer and flushes it if it exceeds the capacity.
  - **buffer_flush** : Writes buffered data to the file and resets the buffer size. Logs a message when the buffer is flushed.
- Control permission
  - The program **retrieves the current user's UID** and checks if the user has the necessary **permissions** to access a specified file.
  - It uses the **getpwuid** function to get user information and **enforces an access control policy** by allowing only a specific user (e.g., "authorized_user") to access the file. This can be modified to check for specific groups or other criteria based on the access control requirements.
- Rate Limiter
  - The program implements a **rate limiter** for file operations, specifically **open and read**, to ensure that no more than **a specified number of operations occur within a given time window**.
  - Can help in preventing resource exhaustion and malicious file operations, ensuring policy compliance, and managing shared resources.

*All integrations to be done with master wrapper

Memory Wrappers

Raadhes, Mith, Dev, Rohan

- brk system call
  - Used to dynamically allocate small memory chunks to a process, for malloc etc.
  - Logging of brk system calls for debugging
  - Ensuring thread safety in cases where glibc ptmalloc2 is not used
- debug malloc
  - Allocates memory and stores pointer and size in a linked list for tracking (AllocRecord).
  - Logs the size and address of each allocation for debugging.
  - Detects and logs errors when allocation fails due to insufficient memory, warns and skips zero size allocation to prevent any undefined behaviour.
  - Helps trace memory usage patterns and provides function to detect memory leaks.
- debug free
  - Verifies that the pointer being freed is valid and tracked.
  - Logs the size and address of the freed memory to confirm deallocation and warns against misuse like double-free or freeing untracked pointers.
  - Removes the pointer from the tracking list after freeing, hence safeguards against errors like freeing invalid memory or using freed pointers.

Memory Wrappers

Raadhes, Mith, Dev, Rohan

\

- Shared Memory :- Wrappers for the commonly used shared memory functions. Created logger libraries for shmget, shmat, shmdt, shctl. Created wrappers that output an error message for the programmer to understand where the error is coming from.
  - shmget wrapper : Creates shared memory segment.
  - shmat wrapper : Attaches memory segment.
  - shmdt wrapper : Detaches memory segment.
  - shmctl wrapper : Destroys memory segment.
  - set_permissions : Uses shmctl to set the permission of the shared memory.